## Testing Model Driven Forms

**Learning Objectives**

* How to setup the TestBed in order to work with forms.
* How to create test specs that test form validity.
* How to test form submission.

**Test setup**

To test forms we’ll extend the LoginComponent we’ve been working with so far in this section. However we’ll convert the simple email, password field form into a model driven form and we’ll drop the input enabled property.

@Component({

selector: 'app-login',

template: `

<

form (ngSubmit)="login()"

①

[formGroup]="form">

②

<label>Email</label>

<input type="email"

formControlName="email">

③

<label>Password</label>

<input type="password"

formControlName="password">

③

<button type="submit">Login</button>

<

/form

>

`

})

export class LoginComponent {

@Output() loggedIn = new EventEmitter<User>();

form: FormGroup;

constructor(private fb: FormBuilder) {

}

ngOnInit() {

④

this.form = this.fb.group({

email: ['', [

Validators.required,

Validators.pattern("[^ @]\*@[^ @]\*")]],

password: ['', [

Validators.required,

Validators.minLength(8)]],

});

}

login() {

console.log(`Login ${this.form.value}`);

if (this.form.valid) {

this.loggedIn.emit(

new User(

this.form.value.email,

this.form.value.password

)

);

}

}

}

1. When the user submits the form we call the login() function.
2. We associate this template form element with the model form on our component.
3. We link specific template form controls to FormControls on our form model.
4. We initialise our form model in the ngOnInit lifecycle hook.

The rest of the LoginComponent looks the same as before.

Our test suite looks mostly the same but has a few key differences:

describe('Component: Login', () => {

let component: LoginComponent;

let fixture: ComponentFixture<LoginComponent>;

beforeEach(() => {

TestBed.configureTestingModule({

imports: [ReactiveFormsModule, FormsModule],

①

declarations: [LoginComponent]

});

// create component and test fixture

fixture = TestBed.createComponent(LoginComponent);

// get test component from the fixture

component = fixture.componentInstance;

component.ngOnInit();

②

});

})

;

1. We add the required ReactiveFormsModule and FormsModule to our test beds imports list.
2. We manually trigger the ngOnInit lifecycle function on our component, Angular won’t call this for us.

Now lets look at how we can test our forms validity.

**Form validity**

The first test spec we may want to check is that a blank form is invalid. Since we are using model driven forms we can just check the valid property on the form model itself, like so:

it('form invalid when empty', () => {

expect(component.form.valid).toBeFalsy();

})

;

We can easily check to see if the form is *valid* by checking the value of component.form.valid.

This one of the reasons model driven forms are easier to test than template driven forms, we already have an object on the component we can inspect from our test spec for *correctness*.

With template driven forms the state is in the *view* and unless the component has a reference to the template form with a ViewChild decorator there is no way to test the form using a unit test. We would have to perform a full E2E test simulating button clicks and typing in values into forms.

**Field validity**

We can also check to see if individual fields are valid, for example the email field should initially be invalid.

it('email field validity', () => {

let email = component.form.controls['email'];

①

expect(email.valid).toBeFalsy();

②

})

;

1. We grab a reference to the actual field itself from the form.controls property.
2. Just like the form we can check if the field is valid through email.valid.

### Field errors

As well as checking to see if the field is valid we can also see what specific validators are failing through the email.errors property.

Since it’s required and the email field hasn’t been set I would expect the requrired validator to be failing, we can test for this like so:

it('email field validity', () => {

let errors = {};

let email = component.form.controls['email'];

errors = email.errors || {};

expect(errors['required']).toBeTruthy();

①

});

① Because errors contains a key of required and this has a *value* this means the required validator is *failing* as we expect.

We can set some data on our input control by calling setValue(…) like so:

email.setValue("test");

If we did set the email field to be test this should fail the pattern validator, since that expects the email to contain a @. We can then check to see if the pattern validator is failing like so:

email.setValue("test");

errors = email.errors || {};

expect(errors['pattern']).toBeTruthy();

**Submitting a form**

We can submit a form by clicking on the submit button, but we’ve already covered this in a previous lecture.

Since the ngSubmit directive has it’s own set of tests it’s safe to *assume* that the (ngSubmit)=login() expression is working as expected.

So to test form submission with model driven forms we can just call the login() function on our controller, like so:

component.login();

Since our form emits an event from the loggedIn output event property we can use the same method we covered in the section on testing components to test this form submission. Namely subscribe to the observable and store a reference to the emitted event for later comparison, like so:

it('submitting a form emits a user', () => {

expect(component.form.valid).toBeFalsy();

component.form.controls['email'].setValue("test@test.com");

component.form.controls['password'].setValue("123456789");

expect(component.form.valid).toBeTruthy();

let user: User; // Subscribe to the Observable and store the user in a local variable.

component.loggedIn.subscribe((value) => user = value);

// Trigger the login function

component.login();

// Now we can check to make sure the emitted value is correct

expect(user.email).toBe("test@test.com");

expect(user.password).toBe("123456789");

});

**Summary**

We can easily unit test model driven forms in Angular by just testing the form model itself.

To test template driven forms in Angular we need to launch a full end to end testing environment and interact with a browser to test the form.

Next we will take a look at how to test an applications that makes http requests.

**Listing**

<http://plnkr.co/edit/FrVMMaLc0NQkArGUC8yb?p=preview>

*login.component.ts*

import {

Component,

EventEmitter,

Output

}

from '@angular/core';

import {

FormGroup,

Validators,

FormBuilder

}

from "@angular/forms";

export class User {

constructor(public email: string,

public password: string) {

}

}

@Component({

selector: 'app-login',

template: `

form (ngSubmit)="login()"

<

[formGroup]="form">

<label>Email</label>

<input type="email"

formControlName="email">

<label>Password</label>

<input type="password"

formControlName="password">

<button type="submit">Login</button>

<

/form

>

`

})

export class LoginComponent {

@Output() loggedIn = new EventEmitter<User>();

form: FormGroup;

constructor(private fb: FormBuilder) {

}

ngOnInit() {

this.form = this.fb.group({

email: ['', [

Validators.required,

Validators.pattern("[^ @]\*@[^ @]\*")]],

password: ['', [

Validators.required,

Validators.minLength(8)]],

});

}

login() {

console.log(`Login ${this.form.value}`);

if (this.form.valid) {

this.loggedIn.emit(

new User(

this.form.value.email,

this.form.value.password

)

);

}

}

}

*login.component.spec.ts*

/\* tslint:disable:no-unused-variable \*/

import {TestBed, ComponentFixture} from '@angular/core/testing'; import {ReactiveFormsModule, FormsModule} from "@angular/forms"; import {LoginComponent, User} from "./login.component";

describe('Component: Login', () => {

let component: LoginComponent;

let fixture: ComponentFixture<LoginComponent>; beforeEach(() => {

// refine the test module by declaring the test component

TestBed.configureTestingModule({

imports: [ReactiveFormsModule, FormsModule],

declarations: [LoginComponent]

});

// create component and test fixture

fixture = TestBed.createComponent(LoginComponent);

// get test component from the fixture

component = fixture.componentInstance;

component.ngOnInit();

});

it('form invalid when empty', () => { expect(component.form.valid).toBeFalsy();

});

it('email field validity', () => {

let errors = {};

let email = component.form.controls['email'];

expect(email.valid).toBeFalsy();

// Email field is required

errors = email.errors || {};

expect(errors['required']).toBeTruthy();

// Set email to something

email.setValue("test");

errors = email.errors || {};

expect(errors['required']).toBeFalsy();

expect(errors['pattern']).toBeTruthy();

// Set email to something correct

email.setValue("test@example.com");

errors = email.errors || {};

expect(errors['required']).toBeFalsy();

expect(errors['pattern']).toBeFalsy();

});

it('password field validity', () => {

let errors = {};

let password = component.form.controls['password'];

// Email field is required

errors = password.errors || {};

expect(errors['required']).toBeTruthy();

// Set email to something

password.setValue("123456");

errors = password.errors || {};

expect(errors['required']).toBeFalsy();

expect(errors['minlength']).toBeTruthy();

// Set email to something correct

password.setValue("123456789");

errors = password.errors || {};

expect(errors['required']).toBeFalsy();

expect(errors['minlength']).toBeFalsy();

});

it('submitting a form emits a user', () => {

expect(component.form.valid).toBeFalsy();

component.form.controls['email'].setValue("test@test.com");

component.form.controls['password'].setValue("123456789");

expect(component.form.valid).toBeTruthy();

let user: User;

// Subscribe to the Observable and store the user in a local variable.

component.loggedIn.subscribe((value) => user = value);

// Trigger the login function

component.login();

// Now we can check to make sure the emitted value is correct

expect(user.email).toBe("test@test.com");

expect(user.password).toBe("123456789");

});

})

;